

Subgraph enumeration in massive graphs*

Francesco Silvestri

IT University of Copenhagen, Denmark
fras@itu.dk

Abstract

We consider the problem of enumerating all instances of a given pattern graph in a large data graph. Our focus is on determining the input/output (I/O) complexity of this problem. Let E be the number of edges in the data graph, $k = O(1)$ be the number of vertices in the pattern graph, B be the block length, and M be the main memory size. The main results of the paper are two algorithms that enumerate all instances of the pattern graph. The first one is a deterministic algorithm that exploits a suitable independent set of the pattern graph of size $1 \leq s \leq k/2$ and requires $O(E^{k-s}/(BM^{k-s-1}))$ I/Os. The second algorithm is a randomized algorithm that enumerates all instances in $O(E^{k/2}/(BM^{k/2-1}))$ expected I/Os; the same bound also applies with high probability under some assumptions. A lower bound shows that the deterministic algorithm is optimal for some pattern graphs with $s = k/2$ (e.g., paths and cycles of even length, meshes of even side), while the randomized algorithm is optimal for a wide class of pattern graphs, called Alon class (e.g., cliques, cycles and every graph with a perfect matching).

1 Introduction

This paper targets the problem of enumerating all subgraphs of an input *data graph* that are isomorphic to a given *pattern graph*. Subgraph enumeration is a tool for analyzing the structural and functional properties of networks (see, e.g., [1, 2]), and typical pattern graphs are cliques (e.g., triangles), cycles and paths. Subgraph enumeration is also strictly related to the evaluation of conjunctive queries or multiway joins on a single large relation [3].

The aim of this paper is to assess the input/output (I/O) complexity of the enumeration problem when the data graph does not fit in the main memory. The main results of the paper are external memory (EM) algorithms for subgraph enumeration. In particular, we provide a deterministic algorithm which exploits a *matched independent set* (MIS) of the pattern graph H , which is an independent set S such that each vertex in S can be matched with a vertex not in S . Let E be the number of edges in the input data graph, $k = O(1)$ be the number of vertices in the pattern graph, B be the block length, and M be the main memory size. Our results are the following:

1. We give a deterministic algorithm for subgraph enumeration that exploits a MIS S of the pattern graph of size $s = |S|$, with $1 \leq s \leq k/2$. Its I/O complexity is $O((E^{k-s} \log_M E)/(BM^{k-s-1}))$. As an example, let $M = \Omega(E^\epsilon)$ for some constant $\epsilon > 0$: we get $O(E^{k-1}/(BM^{k-2}))$ I/Os if the pattern graph is a k -clique ($s = 1$), and $O(E^{k/2}/(BM^{k/2-1}))$ I/Os if the pattern graph is an even length path or cycle, or a mesh of even side ($s = k/2$).

*Part of this work was done while the author was working at the University of Padova. It is supported in part by University of Padova project CPDA121378, MIUR of Italy project AMANDA, and by the European Research Council grant 614331.

2. We propose a randomized algorithm for subgraph enumeration. It exploits the random coloring technique in [4] for decomposing the problem into smaller subproblems that are solved with the above deterministic algorithm. Its expected I/O complexity is $O(E^{k/2}/(BM^{k/2-1}))$. We show that the claimed I/O complexity is also achieved with high probability when $M = \Omega(\sqrt{E} \log E)$ by adjusting the coloring process. We remark that the deterministic algorithm is a crucial component of the randomized one, and cannot be replaced by state-of-the-art techniques without increasing the I/O complexity.
3. We discuss some related issues. We first show that the enumeration of T instances of a pattern graph in the Alon class [5] requires, even in the best case, $\Omega(T/(BM^{k/2-1}) + T^{2/k}/B)$ I/Os. The Alon class includes important graphs like cliques, cycles and, more in general, every graph with a perfect matching. This lower bound implies that the randomized algorithm is optimal in the worst case since a clique with \sqrt{E} vertices contains $T = \Theta(E^{k/2})$ instances of any pattern graph. It also shows that the deterministic algorithm is optimal for some sparse pattern graphs (e.g., even length paths and cycles, meshes of even side) if $M = \Omega(E^\epsilon)$ for some constant $\epsilon > 0$. Finally, we analyze the work complexity of our algorithms: for pattern graphs in the Alon class, the deterministic and randomized algorithms require respectively $\tilde{O}(E^{k-s}/M^{k/2-s})$ and $\tilde{O}(E^{k/2})$ total work, where the last term is just a polylog factor from the optimal bound.

The assumption $k = O(1)$ is quite natural since it covers the most relevant case; however, the analyses of our algorithms do not assume k to be constant and clearly state the dependency of the I/O complexities on k . Moreover, this paper focuses on the enumeration of edge-induced subgraphs which are isomorphic to the pattern graph; however, we claim that our algorithms can be extended even to the enumeration of vertex-induced subgraphs (see Appendix 7.3 for more details).

We do not require our algorithms to *list* all instances of the pattern graph, that is to store all instances on the external memory. We simply consider algorithms that *enumerate* instances: that is, for each instance, they call a function `emit(.)` with the instance as input parameter. Nevertheless, our upper and lower bounds can be easily adapted to list all instances by increasing the I/O complexity of an unavoidable additive $\Theta(T/B)$ factor, where T is the number of instances.

2 Related work and comparison with our results

To the best of our knowledge, this is the first paper to deal with the I/O complexity of the enumeration of a generic pattern graph. Previous works have targeted the I/O complexity of triangle enumeration. An optimal algorithm requiring $O(\text{sort}(E))$ I/Os for graphs with constant arboricity is given in [6]; this algorithm however does not efficiently scale with larger arboricity. The works [7, 8] propose algorithms for a generic data graph incurring $O(E^2/(BM))$ I/Os. In the special case where the pattern graph is a triangle, our deterministic algorithm recalls the one proposed in [8], but it does not need to manage in a different way vertices of the data graph with degree $\leq M$ and with degree $> M$. The previous bound is improved to an optimal $\Theta(E^{3/2}/(B\sqrt{M}))$ (expected) I/O complexity in [4, 9], which respectively provide randomized and deterministic algorithms. Our randomized algorithm extends to a generic pattern graph the random vertex coloring technique introduced in [4]. However, this paper substantially differs from [4] since novel and non-trivial results are proposed: besides specific technicalities required for the generalization of the coloring technique, we give the new deterministic algorithm based on a MIS, which is crucial for solving small subproblems generated by the coloring technique, and we show that the I/O complexity of the randomized algorithm holds even with high probability.

Namely, instances are edge-induced subgraphs of G , while induced instances are vertex-induced subgraphs of G . For a given instance we say that vertex h_i (resp., edge (h_i, h_j)) is *mapped* onto v_i (resp., (v_i, v_j)). An instance is enumerated by calling a function `emit`(v_1, \dots, v_k), and each call performs no I/Os and requires $O(1)$ operations.

We define a *matched independent set* S (MIS) of the pattern graph H to be an independent set of H for which exists in E_H a matching between the s vertices in S and s vertices in $V_H \setminus S$, with $s = |S|$. We have $1 \leq s \leq k/2$. The maximum size of a MIS is $s = \lfloor k/2 \rfloor$ for a cycle of length k or a mesh of size $\sqrt{k} \times \sqrt{k}$, while it is $s = 1$ for a k -clique. For a given MIS S , we let h_{k-s+1}, \dots, h_k denote the vertices of H in S and assume that $h_i \in V_H \setminus S$ is matched with $h_{k-s+i} \in S$ for every $1 \leq i \leq s$. Finally, we define the *probe vertex* of vertex h_i , with $1 \leq i \leq k - s$ as follows: it is h_{k-s+i} if $1 \leq i \leq s$ (i.e., a vertex in S is the probe vertex of its companion in the matching); otherwise it is an arbitrary neighbor vertex in S if $s + 1 \leq i \leq k - s$. If h_j is the probe vertex of h_i , then we say that the *probe index* of i , denoted with $P(i)$, is j and that the *probe edge* of h_i is (h_i, h_j) (see example in Figure 1). Since we are interested in pattern graphs with a very small number of nodes, we suppose that an exhaustive search on the pattern graph is used to find a MIS with the largest size; we leave as open problem to derive an efficient algorithm for extracting a large MIS.

4 Deterministic EM Algorithm

In this section we describe the deterministic algorithm for enumerating all instances of the pattern graph H by exploiting a MIS S of H . The algorithm works for any S , however the best performance are reached when S is the maximum MIS. For the sake of simplicity, we assume that $s < k/2$, and hence that there exists at least one vertex in $V_H \setminus S$, say h_{k-s} , not matched with a vertex in S . The case $s = k/2$, covered in Appendix 7.1, is based on the same approach but requires some minor technicalities that increase the I/O complexity by a multiplicative factor $O(\log_M E)$. This factor is asymptotically negligible as soon as $M = \Omega(E^\epsilon)$ for some constant $\epsilon > 0$. We first provide a simple high level explanation of the algorithm, and then give a more detailed description.

We observe that an instance of H in G is uniquely defined by the mapping of the $k - s - 1$ probe edges associated with h_1, \dots, h_{k-s-1} and of vertex h_{k-s} , since such a mapping automatically fixes the mapping of all vertexes of H . As an example consider again Figure 1: any instance of the pattern graph is univocally given by the mapping of the probe edges $(h_1, h_6), (h_2, h_7), (h_3, h_8), (h_4, h_9)$ and of vertex h_5 . The opposite direction is not true: a mapping may not denote an instance of H since a non-probe edge of H may be mapped on an edge not in G . The deterministic algorithm exploits these facts: it generate all mappings of $k - s - 1$ probe edges and of vertex h_{k-s} , and then verifies which mappings denote real instances of H in G . The generation of all mappings is done with an I/O-efficient exhaustive search.

We assume that the edges of G are split into $\phi = \Theta(Ek/M)$ chunks. Specifically, the adjacency lists of G are split into ϕ consecutive chunks C_i of size in the range $(M/(8k), M/(4k)]$, where $1 \leq i \leq \phi$ and $\phi \in [4kE/M, 8kE/M)$. A vertex whose adjacency list is completely contained in a chunk is called *complete*, and *incomplete* otherwise. We require each chunk to contain at most one incomplete vertex. It can be proved that such a partition exists and can be constructed by scanning the edge set E .

The algorithm works in ϕ^{k-s-1} *rounds*, which run over all possible ways of selecting (with repetitions) $k - s - 1$ chunks from ϕ chunks. In each round, the following operations are done (step numbers refer to the pseudocode in the next page): the $k - s - 1$ selected chunks are loaded into internal memory (steps 1-2); by scanning the entire edge list of G , all edges connecting two incomplete vertexes of the loaded chunks are inserted in memory, if not already in a chunk (step

3); finally, all instances of H where the i -th probe edge is mapped on an edge in the i -th chunk are enumerated (step 4). This last operation proceeds in *iterations* that run over all possible ways of mapping h_{k-s} to a vertex $v \in V$ (note that this vertex is not fixed by the mapping of probe edges). In each iteration (steps 4.a-4.c), the algorithm scans the adjacency list of v and checks if there exists an instance where h_{k-s} is mapped on v and the i -th probe edge is mapped on an edge in the i -th chunk; function `emit(\cdot)` is called for each existing instance.

We now provide a more detailed description of the deterministic algorithm. Consider a generic round and denote with $C_{\ell_1}, \dots, C_{\ell_{k-s-1}}$, for suitable values of $\ell_1, \dots, \ell_{k-s-1}$, the $k-s-1$ selected chunks. The algorithm uses the support sets E', E'', E_i for each $1 \leq i \leq k-s-1$, and V_i for each $1 \leq i \leq k$, which we suppose to be stored in internal memory and initially empty. Each round performs the following operations:

1. For each $1 \leq i \leq k-s-1$, we load in memory C_{ℓ_i} , and fill V_i and E_i with the vertexes and edges that are contained in C_{ℓ_i} . Specifically, we add to V_i all vertexes whose adjacency list is (partially) contained in C_{ℓ_i} , and add to E_i all edges (u, v) where $u \in V_i$ and (u, v) appears in the (part of) adjacency list of u in C_{ℓ_i} .
2. For each $1 \leq i \leq s$, we add to V_{k-s+i} all vertexes of G on which $h_{k-s+i} \in S$ can be mapped assuming that the probe edge of h_i is mapped onto an edge in E_i . Formally, each vertex $u \in V$ is added to V_{k-s+i} if and only if there exists a vertex $v \in V_i$ such that $(v, u) \in E_i$. No I/Os are needed in this step since the operation can be performed by reading the chunks in internal memory.¹
3. Edge set E' is filled with all edges of G connecting vertices in $(\cup_{i=1}^{k-s-1} V_i) \cup (\cup_{i=k-s+1}^k V_i)$ that are not already available in internal memory but are required for correctly enumerating instances. Formally, for each $(h_i, h_j) \in E_H$ with $1 \leq i, j \leq k$ and $i, j \neq k-s$, each edge $(v, v') \in E$ is added to E' if and only if $v \in V_i$, $v' \in V_j$, but $(v, v') \notin E_i \cup E_j$. (We note that an edge can be added to E' although it is contained in E_l for some $l \neq i, j$.) This operation can be performed by scanning once the adjacency lists of G .
4. Enumerate all instances of H in G where vertex h_i is mapped onto a vertex in V_i and its probe edge onto edges in E_i , for any $0 \leq i < k-s$. The enumeration proceeds in V iterations. In an iteration, we set $V_{k-s} = \{v\}$, for any possible value of $v \in V$, and then the following operations are done:
 - (a) Let E'' be the edge set containing all edges between v and vertices in $(\cup_{i=1}^{k-s-1} V_i) \cup (\cup_{i=k-s+1}^k V_i)$ which are not already in internal memory. Formally, each edge (v, v') is added to E'' if and only if $v' \in V_i$ but $(v, v') \notin E_i$. (We note that an edge can be added to E'' although it is contained in E_l for some $l \neq i$.) This step requires a scan of the adjacency list of v .
 - (b) Using a naive approach (see Section 6), enumerate in main memory all instances of H in the subgraph $(\cup_{i=1}^k V_i, E' \cup E'' \cup (\cup_{i=1}^{k-s-1} E_i))$ of G where vertex v_{k-s} is mapped onto v , and the probe edge of h_i is mapped onto an edge in E_i for each $1 \leq i \leq k-s-1$.
 - (c) Empty sets V_{k-s} and E'' .
5. Empty sets E', V_i for each $1 \leq i \leq k$, and E_i for each $1 \leq i \leq k-s-1$.

Correctness and I/O complexity are stated in the following theorem:

Theorem 1. *The above algorithm correctly enumerates all instances of a given pattern graph H and its I/O complexity is $O\left((8k)^{k-s-1} \frac{E^{k-s}}{BM^{k-s-1}}\right)$.*

¹Note that at this point all sets V_i , with $i \neq k-s$, are not empty because $s < k/2$. Indeed, when $s = k/2$, V_k is not filled since h_k is the probe vertex of $h_{k/2}$.

Proof. (Sketch) In order to prove the correctness of the algorithm, it is necessary to prove that all instances are emitted once. As already mentioned all instances are uniquely defined by the mapping of the probe edges of h_1, \dots, h_{k-s-1} and of the vertex h_{k-s} . Standard combinatorial arguments show that each one of these mappings is generated once during the execution of the algorithm. The scan of E performed at the beginning of each round and the scanning of the adjacency list of vertex v at the beginning of an iteration, guarantee that all edges necessary for verifying that a mapping gives a correct instance of H in G are available in the internal memory. The amount of internal memory used in each round is at most M since there are $k-s-1$ chunks of size at most $M/(4k)$ and at most $O(k^2)$ edges are added in steps 3 and 4.a. The naive enumeration in step 4.b then does not require any I/O. The I/O complexity of each round is therefore dominated by the two scans of the adjacency lists of E (in step 3, and in the V iterations of step 4.a). Since there are $\phi^{k-s-1} \leq (8kE/M)^{k-s-1}$ rounds, the claim follows. \square

Proof. We first prove the correctness of the algorithm. Consider an instance (v_1, \dots, v_k) of the pattern graph H in G . For each $1 \leq i \leq k-s-1$, let C_{ℓ_i} be the chunk containing $(v_i, v_{P(i)})$ with $v_i \in V_i$ (we recall that $P(i)$ is the probe index of i , that is $h_{P(i)}$ is the probe vertex of h_i). Consider the unique round where chunks $C_{\ell_1}, \dots, C_{\ell_{k-s-1}}$ are loaded in memory in this order. Then, (v_1, \dots, v_k) is correctly enumerated in the iteration where V_{k-s} is set to v_{k-s} . Indeed, all vertices and edges are available in internal memory: Step 1 guarantees that $v_i \in V_i$ for $1 \leq i \leq k-s-1$; Step 2 adds v_{k-s+i} to V_{k-s+i} for $1 \leq i \leq s$ since the edge $(v_i, v_{k-s+i}) \in E_i$ by assumption and $P(i) = k-s+i$ (we note that this would not happen for V_k when $s = k/2$ since $V_{k/2}$ is empty at this point); Step 3 we have that all edges connecting vertices in $\{v_1, \dots, v_{k-s-1}, v_{k-s+1}, \dots, v_k\}$ are in memory (more specifically, all edges between complete vertices are already in internal memory after Step 1); finally, Step 4a guarantees that all edges between v_{k-s} and $\{v_1, \dots, v_{k-s-1}, v_{k-s+1} \dots v_k\}$ are in memory. The instance (v_1, \dots, v_k) is enumerated once: indeed, the instance can be enumerated only in the unique round where chunks $C_{\ell_1}, \dots, C_{\ell_{k-s-1}}$ are loaded in memory in this order (a different order may enumerate an automorphism but not the same instance), and in the unique iteration where V_{k-s} is set to v_{k-s} (clearly, the naive approach for enumeration in Step 4b must emit each instance once).

We now show that the total amount of required internal memory is at most M . The sets V_i and E_i , for each $i \neq k-s$, have sizes at most $M/4k$ each, and thus at most $M(k-1)/(2k)$ memory words are required (note that chunks $C_{\ell_1}, \dots, C_{\ell_{k-s-1}}$ can be removed from the internal memory after Step 1). The size of V_{k-s} is clearly one memory word. The size of E' is at most $(k-1)^2$ words: indeed, an edge $(v, v') \in E$ is added to E' if and only if $v \in V_i$, $v' \in V_j$, and $(v, v') \notin E_i \cup E_j$; this implies that v and v' are incomplete vertices, otherwise (v, v') would be in $E_i \cup E_j$; then, being at most one incomplete vertex per chunk, the claim follows. Similarly, we have that E'' has size at most $(k-1)$ words. Then, the total amount of space is $M(k-1)/(2k) + k^2$ which is not larger than M since $k \ll M$.

Finally, we analyze the I/O complexity of the algorithm. The I/O cost for enumerating instances in Step 4b is negligible since the problem fits in memory and all operations are performed in main memory. Then the I/O complexity of each round is asymptotically upper bounded by a constant number of scans of the whole edge set E . Since there are $\phi^{k-s-1} \leq (8kE/M)^{k-s-1}$ rounds, the claimed I/O complexity follows. \square

5 Randomized EM Algorithm

We are now ready to introduce the randomized algorithm. The algorithm, by making use of the random coloring technique in [4], decomposes the problem into small subproblems of expected size $O(M)$, which are then solved with the previous deterministic algorithm. We assume that the maximum degree of G is \sqrt{EM} ; however, in Section 5.2, we show how this assumption can

be removed by increasing the I/O complexity by a multiplicative factor $k^{O(k)}$. We first prove the expected I/O complexity and then show how to get the high probability under some assumptions in Section 5.1.

Let $\xi : V \rightarrow \{1, \dots, c\}$, with $c = \sqrt{E/M}$, be a vertex coloring chosen uniformly at random from a family of $2(k-s+1)$ -wise independent family of functions. The coloring ξ partitions the edge set E into c^2 sets of expected size M . For each pair of colors $\tau_1, \tau_2 \in \{1, \dots, c\}$ and $\tau_1 \leq \tau_2$, we denote with E_{τ_1, τ_2} the set containing edges colored with τ_1 and τ_2 , that is $E_{\tau_1, \tau_2} = \{(u, v) \in E \mid \min\{\xi(u), \xi(v)\} = \tau_1, \max\{\xi(u), \xi(v)\} = \tau_2\}$. Each instance (v_1, \dots, v_k) of the pattern graph can be colored by ξ in c^k ways, and it is said to be (τ_1, \dots, τ_k) -colored if $\xi(v_i) = \tau_i$ for each $1 \leq i \leq k$.

The randomized algorithm enumerates all instances by decomposing the problem into c^k subproblems. Each subproblem finds all (τ_1, \dots, τ_k) -colored instances according to a given k -tuple of colors using the previous deterministic algorithm on the edge set $\cup_{\tau_i \leq \tau_j} E_{\tau_i, \tau_j}$. The algorithm is organized as follows:

1. Randomly select a coloring ξ from a $2(k-s+1)$ -wise independent family of functions.
2. Using sorting, store edges in E_{τ_1, τ_2} in consecutive positions, for each color pair (τ_1, τ_2) .
3. For each k -tuple of colors (τ_1, \dots, τ_k) , enumerate all (τ_1, \dots, τ_k) -colored instances using the algorithm in Section 4 on the sets E_{τ_i, τ_j} , for each $\tau_i \leq \tau_j$.

In order to bound the I/O complexity of the randomized algorithm, we introduce the following technical lemma that upper bounds the expected number X_t of possible tuples of t edges in E that are colored in the same way by ξ . A closed form of this quantity is $X_t = \sum_{\tau_1 \leq \tau_2, E_{\tau_1, \tau_2} \geq t} \frac{E_{\tau_1, \tau_2}!}{(E_{\tau_1, \tau_2} - t)!}$ (note that sets E_{τ_1, τ_2} with less than t edges do not contribute).

Lemma 1. *Let $\xi : V \rightarrow \{1, \dots, c\}$ be chosen uniformly at random from a $2t$ -wise independent family of hash functions, where $c = \sqrt{E/M}$. If $M = \Omega(t^2)$ and the maximum vertex degree in G is \sqrt{EM} , then $\mathbb{E}[X_t] \leq (2t)^{t-1} EM^{t-1}$.*

Proof. We prove the claim by induction on t . The claim is verified for $t = 1$ since $\mathbb{E}[X_1] = E$. For each tuple $\mathbf{e} = (e_1, \dots, e_t)$ of t distinct edges in E and for each $2 \leq i \leq t$, let $Y_i^{\mathbf{e}} = 1$ if e_i is in the same set E_{τ_1, τ_2} , for some colors τ_1, τ_2 , of edges e_1, \dots, e_{i-1} , and 0 otherwise. Set $Y_1^{\mathbf{e}} = 1$. We get $X_t = \sum_{\mathbf{e}} Y_t^{\mathbf{e}}$. Since there are at most $2t$ vertices and ξ is $2t$ -wise, we get

$$\Pr(Y_t^{\mathbf{e}} = 1) \leq \begin{cases} \Pr(Y_{t-1}^{\mathbf{e}} = 1) / c^2 & \text{if } e_t \text{ is not adjacent to } e_1, \dots, e_{t-1} \\ \Pr(Y_{t-1}^{\mathbf{e}} = 1) / c & \text{if } e_t \text{ is adjacent to } e_1, \dots, e_{t-1} \text{ on one vertex} \\ \Pr(Y_{t-1}^{\mathbf{e}} = 1) & \text{if } e_t \text{ is adjacent to } e_1, \dots, e_{t-1} \text{ on two vertices} \end{cases}$$

Each $(t-1)$ -tuple \mathbf{e}' can be extended by at most E edges that are not connected with \mathbf{e}' , or by $2(t-1)\sqrt{EM}$ edges that are connected to \mathbf{e}' on just one vertex (recall that the maximum degree of a vertex is \sqrt{EM}), or by $(t-1)(2t-3)$ edges that are connected to \mathbf{e}' on two vertices. Therefore, we get

$$\mathbb{E}[X_t] = \sum_{\mathbf{e}} \Pr(Y_t^{\mathbf{e}} = 1) \leq \mathbb{E}[X_{t-1}] \left(\frac{E}{c^2} + 2(t-1) \frac{\sqrt{EM}}{c} + (t-1)(2t-3) \right).$$

Since the right term is upper bounded by $2tM\mathbb{E}[X_{t-1}]$, the lemma follows. \square \square

We are now ready to show the correctness and I/O complexity of the randomized algorithm.

Theorem 2. *The above randomized algorithm enumerates all instances of a given pattern graph H . If the maximum vertex degree of G is \sqrt{EM} , then the expected I/O complexity of the algorithm is $O((8k)^{4(k-s+1)} E^{k/2} / (BM^{k/2-1}))$.*

Proof. The correctness easily follows since each instance is colored with a suitable color tuple (τ_1, \dots, τ_k) and is enumerated only in the subproblem associated with this color tuple. The cost of each subproblem is given by Theorem 1, however for simplicity, we upper bound the cost of the deterministic algorithm with $O((8k)^{k-s} E^{k-s+1} / (BM^{k-s}))$ in order to get rid of the logarithmic term. The I/O complexity $Q(E)$ of the algorithm is upper bounded by the sum of the costs of all c^k subproblems. Then,

$$\begin{aligned}
Q(E) &= O \left(\frac{(8k)^{k-s}}{BM^{k-s}} \sum_{(\tau_1, \dots, \tau_k)} \left(\sum_{\tau_i \leq \tau_j} E_{\tau_i, \tau_j} \right)^{k-s+1} \right) \\
&\leq O \left(\frac{(8k)^{2(k-s+1)}}{BM^{k-s}} \sum_{(\tau_1, \dots, \tau_k)} \sum_{\tau_i \leq \tau_j} E_{\tau_i, \tau_j}^{k-s+1} \right) \\
&\leq O \left(\frac{c^{k-2} (8k)^{2(k-s+1)}}{BM^{k-s}} \sum_{\tau_1 \leq \tau_2} E_{\tau_1, \tau_2}^{k-s+1} \right) \\
&\leq O \left(\frac{c^{k-2} (8k)^{3(k-s+1)}}{BM^{k-s}} \sum_{\tau_1 \leq \tau_2, E_{\tau_1, \tau_2} \geq k-s+1} \frac{E_{\tau_1, \tau_2}!}{(E_{\tau_1, \tau_2} - k + s - 1)!} \right) \\
&\leq O \left(\frac{c^{k-2} (8k)^{3(k-s+1)}}{BM^{k-s}} X_{k-s+1} \right).
\end{aligned}$$

By the linearity of expectation, we get $\mathbb{E}[Q(E)] = O \left(\frac{c^{k-2} (8k)^{3(k-s+1)}}{BM^{k-s}} \mathbb{E}[X_{k-s+1}] \right)$. Then, by Lemma 1 and the $2(k-s+1)$ -wiseness of ξ , we get the claimed result. \square

We remark that our deterministic algorithm is crucial for getting the claimed I/O complexity. Indeed, the algorithm used in the subproblems should require $O(M/B)$ I/Os for solving subproblems of size $\Theta(M)$ (note that subproblems may not perfectly fit the memory size). Using existing enumeration algorithms, which require $\Omega(M^{k/2}/B)$ I/Os for solving subproblems of size $\Theta(M)$, would increase the total I/O complexity by a multiplicative factor $\Omega(M^{k/2-1})$.

5.1 Getting the high probability

If $M = \Omega(\sqrt{E} \log E)$, the randomized coloring process can be slightly modified to get with probability $1 - 1/\Theta(E)$ the claimed I/O complexity. For the sake of simplicity we assume the maximum degree to be \sqrt{EM} , although it is possible to remove this assumption even for higher degree by adapting the procedure described in the next Section 5.2.²

A vertex $v \in V$ has *high degree* if $\sqrt{E} \leq \deg(v) \leq \sqrt{EM}$ and has *low degree* if $\deg(v) < \sqrt{E}$. The coloring process is modified as follows. The colors of low degree vertices are assigned independently and uniformly at random. The colors of high degree vertices are set by partitioning vertices into c groups so that the sum of degrees within each group is in $[\sqrt{EM}, 2\sqrt{EM})$, and then high degree vertices within the i -th group get color i (this operation requires $O(1)$ sorts).

Our argument relies on the technique by Janson [19, Theorem 2.3] for obtaining a strong deviation bound for sums of dependent random variables, which we recall here for completeness. Let $X = \sum_{i=1}^p Y_i$ where each Y_i is a random variable with $Y_i - \mathbb{E}[Y_i] \leq 1$, and let $\psi = \sum_{i=1}^p \text{Var}(Y_i)$. Denote with Δ the maximum degree of the dependency graph of Y_1, \dots, Y_p : this is a graph with vertex set $Y = \{1, \dots, p\}$ such that if $B \subset Y$ and $i \in Y$ is not connected to a vertex

²For $k = O(1)$, the procedure in Section 5.2 consists in repeating the randomized algorithm a constant amount of times. Then by an union bound, we get that the claimed complexity.

in B , then Y_i is independent of $\{Y_j\}_{j \in B}$. Then, for any $d > 0$, we have $\Pr(X \geq (1+d)\mathbb{E}[X]) \leq e^{-\frac{8d^2\mathbb{E}[X]^2}{25\Delta(\psi+d\mathbb{E}[X]/3)}}$.

Theorem 3. Let $M = \Omega(\sqrt{E} \log E)$ and let the maximum vertex degree of G be \sqrt{EM} . Then, the I/O complexity of the above algorithm is $O((8k)^{6(k-s)} E^{k/2} / (BM^{k/2-1}))$ with probability at least $1 - 1/E$.

Proof. Let E^L be the set of edges in E connecting two low degree vertices. We also define $E^H = E/E^L$, $E_{\tau_1, \tau_2}^L = E_{\tau_1, \tau_2} \cap E^L$, $E_{\tau_1, \tau_2}^H = E_{\tau_1, \tau_2} \cap E^H$. We first show that the size of E_{τ_1, τ_2}^L for any color pair τ_1, τ_2 is smaller than $2M$ with probability at least $1 - 1/(2E)$. Assume for simplicity that $|E^L| = |E|$. For each edge $e \in E^L$, define the random variable Y_e to be 1 if edge e is in E_{τ_1, τ_2}^L , and 0 otherwise. We thus have $|E_{\tau_1, \tau_2}^L| = \sum_{e \in E} Y_e$. Each random variable Y_e depends on the at most $2\sqrt{E}$ variables associated with edges adjacent to e , while it is independent of the remaining ones.³ Since $Y_e - \mathbb{E}[Y_e] < 1$, we use the aforementioned result by Janson by setting $p = E$, $\mathbb{E}[Y_e] = M$, $\psi = E(1/c^2 - 1/c^4) < M$, $d = 1$, $\Delta = 2\sqrt{E}$. Then we get $\Pr(|E_{\tau_1, \tau_2}^L| \geq 2M) \leq e^{-\frac{4M}{25\sqrt{E}}}$. By an union bound, the probability that E_{τ_1, τ_2}^L is smaller than $2M$ for every color pair is at least $1 - c^2 e^{-\frac{4M}{25\sqrt{E}}} \geq 1 - 1/(2E)$ when $M = \Omega(\sqrt{E} \log E)$.

We now show that the set E_{τ_1, τ_2}^H has size $8M$ with probability at least $1 - 1/(2E)$. There are at most $2\sqrt{M}$ high degree vertices colored with a given color. Then, there cannot be more than $4M$ edges connecting two high degree vertices in E_{τ_1, τ_2}^H . Consider now the set E^{H*} of edges connecting high degree vertices of colors τ_1 or τ_2 to low degree vertices. We have $|E^{H*}| \leq 4\sqrt{EM}$. For each $e \in E^{H*}$, define the random variable Y_e to be 1 if the low degree vertex gets color τ_1 or τ_2 , and 0 otherwise. We have $|E_{\tau_1, \tau_2}^H| \leq \sum_{e \in E^{H*}} Y_e$. Since random variables may be dependent, we apply again the result by Janson with $p = 4\sqrt{EM}$, $\mathbb{E}[Y_e] = 8M$, $\psi = \sum_{e \in E^{H*}} \text{Var}(Y_e) \leq 8M$, $d = 1/2$, $\Delta = 2\sqrt{E}$ (since only low degree vertices are randomly colored). Then, $\Pr(|E_{\tau_1, \tau_2}^H| \geq 12M) \leq e^{-\frac{2M}{25\sqrt{E}}}$. Then, the probability that E_{τ_1, τ_2}^H is smaller than $16M$ for every color pair is at least $1 - c^2 e^{-\frac{2M}{25\sqrt{E}}} \geq 1 - 1/(2E)$ when $M = \Omega(\sqrt{E} \log E)$.

Therefore, we have that each E_{τ_1, τ_2} has size at most $16M$ with probability at least $1 - 1/E$. Since each subproblem receives at most k^2 edge sets, the I/O complexity of a subproblem is $O((18k^2)^{k-s} (8k)^{4(k-s-1)} M/B)$. Since there are c^k subproblems, the claimed I/O complexity follows. \square

It deserves to be noticed that it is possible to color low degree vertices with a coloring from a $2(k-s)$ -wise independent family and still get the claimed I/O complexity with probability $1 - 1/E^\epsilon$, for $0 \leq \epsilon \leq 1/4$, as soon as $M \geq E^{3/4+\epsilon}$. It suffices to use a technique by Gradwohl and Yehudayoff [20, Corollary 3.2] in our argument instead of the aforementioned result by Janson [19, Theorem 2.3].

5.2 Removing the degree assumption

Although the assumption in the randomized algorithm that the maximum degree in G is at most \sqrt{EM} is reasonable for real datasets, it can be removed by increasing the I/O complexity by a multiplicative $k^{O(k)}$ factor. We use the previous randomized algorithm as a black box and exploit a coloring technique that should not be confused with the one used inside the randomized algorithm. We denote with V_H the set of *very high degree* vertices in G (i.e., degree larger than \sqrt{EM}), and with $V_L = V \setminus V_H$ the remaining low degree vertices. We let $G_L = (V_L, E_L)$ denote the subgraph of G induced by V_L .

³Note that this is not the case if low degree vertices were colored with $2(k-s)$ -wise independent hash functions.

Let $p \in [0, k]$. Consider the following simpler problem: enumerate all instances of H where p given vertices of H , say for notational simplicity h_1, \dots, h_p , are respectively mapped onto p given very high degree vertices v'_1, \dots, v'_p , and where the remaining vertices of H are mapped onto vertices in V_L . Since the mapping on the first p vertices is given we assume that if $(h_i, h_j) \in E_H$ then $(v'_i, v'_j) \in E$ for any $1 \leq i, j \leq p$ (this can be checked in scanning complexity). We now show that this problem reduces to the enumeration in G_L of a suitable *colored* pattern graph with $k' = k - p$ vertices, and which can be solved with the previous randomized algorithm. Suppose that each vertex in V_L is colored with a p -bit color, initially set to 0. Then, for each $i \in [1, p]$ and for each vertex $v \in V_L$ adjacent to v'_i , we update the color of v by setting the i -th bit to 1 (note that at the end of this operation, a vertex color can have several bits set to 1). Define the color tuple $d = (d_1, \dots, d_{k'})$ as follows: set each term to 0; then, for each $1 \leq i \leq p$ and for each h_{p+j} adjacent to h_i in H , we set the i -th bit of d_j to 1. Let H' be the subgraph of H induced by h_{p+1}, \dots, h_k . Then, the problem can be solved by emitting instances $(v'_1, \dots, v'_p, v''_1, \dots, v''_{k'})$, where $(v''_1, \dots, v''_{k'})$ is every instance of H' in G_L where vertices are colored according with d (i.e., the i -th vertex of the instance has color d_i). The colored instances of H' can be obtained by adapting the previous randomized algorithm to throw away instances that are not compatible with coloring d .

By iterating the previous technique for any value of p and for any matching of p vertices in H with p very high degree vertices, we get the claimed result.

Theorem 4. *The above algorithm enumerates all instances of a given pattern graph H and the expected I/O complexity is $O(k^{5(k-s+1)} E^{k/2} / (BM^{k/2-1}))$.*

Proof. We first show that the technique correctly enumerates all instances where h_1, \dots, h_p are respectively mapped onto the very high degree vertices v'_1, \dots, v'_p , and where the remaining vertices of H are mapped onto vertices in V_L . Consider the emitted $(v'_1, \dots, v'_p, v''_1, \dots, v''_{k'})$ tuple. We now prove that this instance satisfies the required properties. Clearly, $v''_i \in V_L$ by construction. We now show that if $(h_i, h_j) \in E_H$ then the edge is correctly mapped onto E . If $1 \leq i, j \leq p$ or $p+1 \leq i, j \leq k$, the claim is verified, respectively, by the initial assumption on v'_1, \dots, v'_p and by the correctness of the randomized algorithm. Suppose $1 \leq i \leq p$ and $p+1 \leq j \leq k$ (the opposite is equivalent). Color d_{j-p} must have the i -th bit set to 1 since $(h_i, h_j) \in E_H$. Since the instance must verify the coloring tuple, vertex v''_{j-p} has color d_{j-p} and then it is adjacent to v'_i since the i -th bit is 1. Vice versa, it can be similarly shown that all instances that satisfy the desired properties are correctly enumerated.

Let $r \leq 2\sqrt{E/M}$ be the number of very high degree vertices. Since for a given p the technique is called $r^p \frac{k!}{(k-p)!}$, the expected I/O complexity can be upper bounded as follows:

$$O\left(\sum_{p=0}^k r^p \frac{k!}{(k-p)!} k^{4(k-p-s+1)} \frac{E^{(k-p)/2}}{BM^{(k-p)/2-1}}\right) = O\left(k^{5(k-s+1)} \frac{E^{k/2}}{BM^{k/2-1}}\right).$$

□

□

We note that the subsequent lower bound does not hold for the technique proposed for getting rid of the degree assumption. Indeed, information on graph connectivity are encoded in the coloring bits, but the lower bound requires at least one memory word for each vertex or edge. However, if k is a small constant, the lower bound still applies by using a memory word instead on a single bit.

6 Further Extensions

6.0.1 Lower Bound on I/O Complexity

We now describe a lower bound on the I/O complexity for any algorithm that enumerates T instances of a pattern graph in the class of graphs named *Alon class* [5]. A graph in the Alon class has the property that vertices can be partitioned into disjoint sets such that the subgraph induced by each partition is either a single edge, or contains an odd-length Hamiltonian cycle. As in previous works [8, 4] on triangle enumeration, we assume that each edge or vertex requires at least one memory word. That is, at any point in time there can be at most M edges/vertices in memory, and an I/O can move at most B edges/vertices to or from memory. This assumption is similar to the indivisibility assumption which is common in lower bounds on the I/O complexity. My mimic the argument in [4] for triangle enumeration, it can be proved that the enumeration requires $\Omega(T/(BM^{k/2-1}) + T^{2/k}/B)$ I/Os. The claim follows by the fact that there cannot be more than $\Theta(m^{k/2})$ instances of a subgraph in the Alon class in a graph of m edges [21]. (For the sake of completeness we provide the proof in Appendix 7.2). When $k = O(1)$, the lower bound shows that our randomized algorithm is optimal for any pattern graph, while the deterministic algorithm is optimal if $s = k/2$ and $M = E^\epsilon$ for some constant $\epsilon > 0$. Indeed, if the data graph is a complete graph with \sqrt{E} vertices, there exist $T = \Theta(E^{k/2})$ instances of any pattern graph with k vertices.

6.0.2 Work Complexity

We analyze the work complexity when the pattern graph is in the Alon class and $k = O(1)$. By using the ideas in [3, Theorem 6.2], the enumeration (in internal memory) within each iteration of the deterministic algorithm can be performed in $\tilde{O}(M^{k/2-1})$ work. Then the total work of the deterministic algorithm is $\tilde{O}(E^{k-s}/M^{k/2-s})$. As a consequence the expected work of the randomized algorithm becomes $\tilde{O}(E^{k/2})$, which is just a polylog factor from the optimum since instances in the Alon class (e.g., cliques) can appear $\Theta(E^{k/2})$ times in the worst case. To the best of our knowledge, the only algorithm for enumerating a generic pattern graph which does not belong to the Alon class is a brute-force approach. In this case, the deterministic algorithm requires $\tilde{O}(E^{k-s})$ work since Step 4b can be performed in $\tilde{O}(M^{k-s-1})$ work using the brute-force approach; the expected work of the randomized algorithm then becomes $\tilde{O}(E^{k/2}M^{k/2-s})$. In this case the work may become the main bottleneck in a practical implementation.

7 Conclusion

The worst case complexities of our algorithms have an exponential dependency on the vertex number k of the pattern graph, and they are thus mainly of theoretical interest. The lower bound shows that this is the best result in the worst case under standard assumptions. However, some experiments [17] on related MapReduce algorithms for triangle enumeration shows interesting performance and seems to suggest that the analysis of our algorithms can be improved by expressing the complexities as function of some properties of the input graph (e.g., arboricity) or of the output. An output sensitive algorithm for triangle enumeration has recently been proposed by Björklund et al. [22] in the RAM model, however the problem remains open in the external memory for the enumeration of an arbitrary subgraph as well as for triangle enumeration.

Acknowledgments. The author would like to thank Rasmus Pagh and Andrea Pietracaprina for useful discussions.

References

- [1] Kairam, S.R., Wang, D.J., Leskovec, J.: The life and death of online groups: Predicting group growth and longevity. In: Proc. 5th WSDM. (2012) 673–682
- [2] Gregori, E., Lenzini, L., Mainardi, S.: Parallel k-clique community detection on large-scale networks. IEEE Trans. Paral. Dist. Systems **24**(8) (2013) 1651–1660
- [3] Afrati, F.N., Fotakis, D., Ullman, J.D.: Enumerating subgraph instances using Map-Reduce. In: Proc. 29th ICDE. (2013) 62–73
- [4] Pagh, R., Silvestri, F.: The input/output complexity of triangle enumeration. In: Proc. of 33rd PODS. (2014) 224–233
- [5] Afrati, F.N., Sarma, A.D., Salihoglu, S., Ullman, J.D.: Upper and lower bounds on the cost of a Map-Reduce computation. Proc. VLDB Endow. **6**(4) (2013) 277–288
- [6] Goodrich, M., Pszona, P.: External-memory network analysis algorithms for naturally sparse graphs. In: Proc. 19th ESA. Volume 6942 of LNCS. (2011) 664–676
- [7] Chu, S., Cheng, J.: Triangle listing in massive networks. ACM Trans. Knowl. Discov. Data **6**(4) (2012) 17:1–17:32
- [8] Hu, X., Tao, Y., Chung, C.W.: Massive graph triangulation. In: Proc. ACM SIGMOD. (2013) 325–336
- [9] Hu, X., Qiao, M., Tao, Y.: Join dependency testing, loomis-whitney join, and triangle enumeration. In: Proc. 34th ACM PODS. (2015) 291–301
- [10] Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. **14**(1) (1985) 210–223
- [11] Ngo, H.Q., Porat, E., Ré, C., Rudra, A.: Worst-case optimal join algorithms. In: Proc. 31st PODS. (2012) 37–48
- [12] Kolountzakis, M.N., Miller, G.L., Peng, R., Tsourakakis, C.E.: Efficient triangle counting in large graphs via degree-based vertex partitioning. Internet Mathematics **8**(1-2) (2012) 161–185
- [13] Kane, D.M., Mehlhorn, K., Sauerwald, T., Sun, H.: Counting arbitrary subgraphs in data streams. In: Proc. 39th ICALP. (2012) 598–609
- [14] Vassilevska Williams, V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. SIAM J. Comput. **42**(3) (2013) 831–854
- [15] Finocchi, I., Finocchi, M., Fusco, E.G.: Counting small cliques in mapreduce. arXiv:1403.0734 (2014)
- [16] Suri, S., Vassilvitskii, S.: Counting triangles and the curse of the last reducer. In: Proc. 20th ACM International Conference on World Wide Web. (2011) 607–614
- [17] Park, H.M., Silvestri, F., Kang, U., Pagh, R.: Mapreduce triangle enumeration with guarantees. In: Proc. 23rd CIKM. (2014)
- [18] Vitter, J.S.: Algorithms and Data Structures for External Memory. Now Publishers Inc. (2008)
- [19] Janson, S.: Large deviations for sums of partly dependent random variables. Random Structures & Algorithms **24**(3) (2004) 234–248
- [20] Gradwohl, R., Yehudayoff, A.: t-wise independence with local dependencies. Information Processing Letters **106**(5) (2008) 208 – 212
- [21] Alon, N.: On the number of subgraphs of prescribed type of graphs with a given number of edges. Israel Journal of Mathematics **38**(1-2) (1981) 116–130
- [22] Björklund, A., Pagh, R., Williams, V.V., Zwick, U.: Listing triangles. In: Proc. of 41st ICALP. Volume 8572 of LNCS. (2014) 223–234

Appendix

7.1 Deterministic EM Algorithm when $s = k/2$

We now explain how to extend the algorithm to the case $s = k/2$, that is when all vertices in $V_H \setminus S$ are matched with a vertex in S . Note that in this case k must be even. We recall that, with our notation, h_i is matched with $h_{k/2+i}$ under S for each $1 \leq i \leq k/2$. Let Γ_k denote the set of indexes of vertices in $V_H \setminus h_{k/2}$ adjacent to h_k (i.e., $i \in \Gamma_k$ if and only if $i < k/2$ and $(h_i, h_k) \in E_H$).

We observe that in the previous algorithm, the vertex set V_k is empty in Step 4b since h_k is the probe vertex of $h_{k/2}$ and thus V_k is not filled in Step 2. If there are no incomplete vertices in each chunk, then the previous algorithm can be fixed by filling V_k in Step 2 with vertices that are connected to a vertex in V_j for every $j \in \Gamma_k$. Indeed, these are the only possible values on which v_k can be mapped when all vertices h_i with $1 \leq i < k/2$ are mapped onto vertices in V_i . This operation requires no I/Os since all adjacency lists in each chunk are completely contained in internal memory, and hence the upper bound in Theorem 1 still applies. Instead of proving this claim, we propose a more general approach that holds even with incomplete vertices.

Two major changes are required in the deterministic algorithm. The first one allows to correctly enumerate all instances where at least one vertex in $\{h_i, \forall i \in \Gamma_k\}$ is mapped onto a complete vertex. Then the second change, which is more articulated, allows to enumerate all instances where each vertex in $\{h_i, \forall i \in \Gamma_k\}$ is mapped onto an incomplete vertex.

First change. In Step 2, we add to V_k all vertices v which are neighbors of complete vertices in V_j for some $j \in \Gamma_k$. Specifically, for each edge (u, v) in E_j with $j \in \Gamma_k$, $u \in V_j$ and u complete, vertex v is added to V_k . As we will see in the main proof, this change allows to enumerate all instance where at least one vertex in $\{h_i, \forall i \in \Gamma_k\}$ is mapped onto a complete vertex. For clearness, consider the following example. Let h_1 be adjacent to h_k and let v be a complete vertex in V_1 . If h_1 is mapped onto v , the possible values onto which h_k can be mapped is given by the adjacency list of v which is totally in memory (thus, V_k is set to these vertices). Then, for complete the enumeration in the current round we have to insert into E' each edge connecting incomplete vertices in each V_j , for $j \in \Gamma_k$ with a vertex in V_k (this operation is performed by Step 3 without further modifications)

Second change. We add a new operation before Step 5, but outside the iteration loop in Step 4. This operation is performed only if there exists an incomplete vertex in each chunk C_{ℓ_i} with $i \in \Gamma_k$ and let $v'_1, \dots, v'_{\Gamma_k}$ these vertices (otherwise, there would no instances where each vertex in $\{h_i, \forall i \in \Gamma_k\}$ is mapped onto an incomplete vertex and this modification would be useless). The algorithm computes a set V' , stored in external memory since it may exceed the internal memory size, containing all vertices that are connected with all vertices $v'_1, \dots, v'_{\Gamma_k}$ in G ; this set can be computed by merging the adjacency lists of $v'_1, \dots, v'_{\Gamma_k}$ and keeping only vertices that appear Γ_k times. Then, using sorting, we compute a new edge list \hat{E} containing all edges with at least one extreme in V' . For each edge in \hat{E} , we call *linked* the vertex in V' . We denote with \hat{V} the vertices in \hat{E} and require \hat{E} to be stored as a collection of adjacency lists. Subsequently, the algorithm enumerates all instances where vertex $h_{k/2}$ is mapped onto a vertex in \hat{V} , its probe edge onto an edge in \hat{E} , h_k onto the linked vertex of this edge (i.e., with a vertex in V'), and h_i on the incomplete vertex in V_i for each $i \in \Gamma_k$. The enumeration is performed in iterations as in the previous algorithm, and in each iteration the algorithm maps $h_{k/2}$ on a vertex \hat{V} and its adjacency list loaded in memory (if the adjacency list is too long we split it into segments of size $M/(8k)$). (Clearly, as for every instance enumerated in the current round, we also require that vertex h_i is mapped onto a vertex in V_i and its probe edge onto E_i for any $1 \leq i \leq k/2 - 1$.) We observe that it is not needed to load in memory the adjacency lists of incomplete vertices in $\{V_i, \forall i \in \Gamma_k\}$ since each vertex in V' is connected with all of them by construction. The I/O complexity of the algorithm is bounded by the following theorem.

Theorem 5. *The above algorithm correctly enumerates all instances of a given pattern graph H and its I/O complexity is $O\left((8k)^{k-s-1} \frac{E^{k-s}}{BM^{k-s-1}} \log_M E\right)$.*

Proof. Consider the first change. In Step 3, we load in memory each edge between incomplete vertices in $\cup_{i \in \Gamma_k} V_i$ and vertices in V_k . By mimic the proof of Theorem 1, it can be shown that the algorithm correctly enumerates all instances where vertex h_i is mapped onto a vertex in V_i , its probe edge onto E_i for any $1 \leq i \leq k/2 - 1$ and at least one vertex in $\{h_i, \forall i \in \Gamma_k\}$ is mapped onto a complete vertex. However, it may happen that instances where all vertices in $\{h_i, \forall i \in \Gamma_k\}$ are mapped onto incomplete vertices are not enumerated since some edges could be missing in E' .

This is fixed by the second change. Indeed, the construction on \hat{E} guarantees that for each $(u, v) \in \hat{E}$, where v is marked as linked, the vertex v is connected to every incomplete vertex in $\{V_i, \forall i \in \Gamma_k\}$. Therefore, as soon as h_i is mapped on the incomplete vertex in V_i , with $i \in \Gamma_k$, and h_k is mapped onto v , we have that the edge dependencies are correctly enumerated (even if edge information are not currently available in internal memory).

Finally, we note that the first change does not increase the I/O complexity and load in memory at most $\Gamma_k \cdot M/(2k) \leq M/4$ additional edges/vertices. The second change requires $O((E/B) \log_M E)$ I/Os per round (i.e., sorting complexity) and load in memory at most $M/(8k)$ edges per iteration. Since the space used by the first change can be deallocated before the operations required by the second change start, the total amount of internal memory never exceeds M (recall as shown in the previous theorem, the base algorithm requires about $M(k-1)/(2k)$ words of internal memory). The claimed I/O complexity easily follows. \square

We observe that the deterministic algorithm requires a MIS S of the pattern graph (i.e., each vertex in S is matched with a vertex not in S) in order to correctly enumerate instances with incomplete vertices. As an example consider the following case. Let the pattern graph be a path of length 3, let h_1 be adjacent to vertices h_0 and h_2 , and let $S = \{h_0, h_2\}$ be a standard independent set of H (note that it is not matched). Suppose that there exists an instance where vertices v, v', v'' are mapped onto h_0, h_1, h_2 respectively. If v' is incomplete and edges (v, v') and (v', v'') are in distinct chunks, then the two edges may not be at the same time in the internal memory and then the instance cannot be emitted. This problem disappears if the maximum degree of the input data graph is $O(M/k)$ since there are no incomplete vertices and then all edges connected to a vertex are available within a single chunk. In this case, it can be proved that the I/O complexity reduces to $O\left((8k)^{k-s'-1} E^{k-s'}/(BM^{k-s'-1})\right)$ I/Os, where s' is the size of a traditional independent set S' of the pattern graph. This implies that it is possible to go below the $O(E^{k/2}/(BM^{k/2-1}))$ bound if $s' \geq k/2$, such as in stars, paths of odd length, or meshed with odd side. Clearly, for these patterns the lower bound in Section 6 does not apply since they are not in the Alon class.

7.2 Lower bound on the I/O Complexity

The proof mimics the argument in [4] for triangle enumeration, but exploits the fact that there cannot be more than $\Theta(m^{k/2})$ instances of a subgraph in the Alon class in a graph of m edges [21]. The execution of an algorithm on a memory of size M can be simulated, without increasing the I/O complexity, in a memory of size $2M$ so that the computation proceeds in rounds. In each round (with the possible exception of the last round) there are $\Theta(M/B)$ I/Os, and memory blocks are read from (resp., written on) the external memory only at the begin (resp., end) of a round. (We refer to [4] for more details on the simulation.) By the aforementioned result on the Alon class, $\Theta(M^{k/2})$ instances can be enumerated in a round since there are at most $2M$ edges in memory. Then, there must be at least $\lfloor T/\Theta(M^{k/2}) \rfloor$ rounds. Since each round needs $\Theta(M/B)$ I/Os, we get the first part of the claim. The second term follows since $\Omega(T^{2/k})$ input edges must be read to enumerate T distinct instances. \square

7.3 Enumeration of Induced Subgraphs.

The deterministic and randomized algorithms can be easily adapted to enumerate all *induced* instances of a given subgraph. The I/O complexity of the deterministic algorithm does increase asymptotically, while the I/O complexity of the randomized algorithm shows only a small increase in the exponent of the term $k^{O(k)}$. It suffices to run the deterministic algorithm as the subgraph was a k -clique $s = 1$ and hence we can use the simple deterministic algorithm bounded in Theorem 1). In each iteration, the algorithm contains all edges in E between any pair of vertices in $\cup_{i=1}^k V_i$. Then, all instances of H are found, but only induced instances are enumerated. This is possible since all edges between vertices in the instance are available in memory. The I/O complexity of the algorithm then becomes $O((8k)^{k-2}E^{k-1}/(BM^{k-2}))$. By using this algorithm for solving subproblems in the randomized algorithm, we get an enumeration algorithm for induced subgraphs requiring $O((8k)^{4k}E^{k/2}/(BM^{k/2-1}))$ I/Os, assuming that the maximum vertex degree is \sqrt{EM} . The high probability result applies as well.